# Linux Netfilter/IPTables Host-Based Firewall Basics

**Jim Guggemos**

**jaguggemos@lbl.gov**

**Unix Systems Engineer**
**Information Technologies & Services Division**

**July 16, 2002**

# Overview

- Internet Networking (TCP, UDP, IP, ICMP)
- What is a Host-Based Firewall?
- Why Should I Use a Host-Based Firewall?
- Firewall Basics
- Stateful (Dynamic) Packet Filtering
- IPTables Basics
- Building a Usable Stand-Alone Firewall Ruleset
- Diagnosing Firewall Problems
- References

# Internet Networking
# Protocol Stack & Sample Connection

## Protocol Stack

**Application Layer**
Client and Server Programs (e.g. HTTP, SMTP)

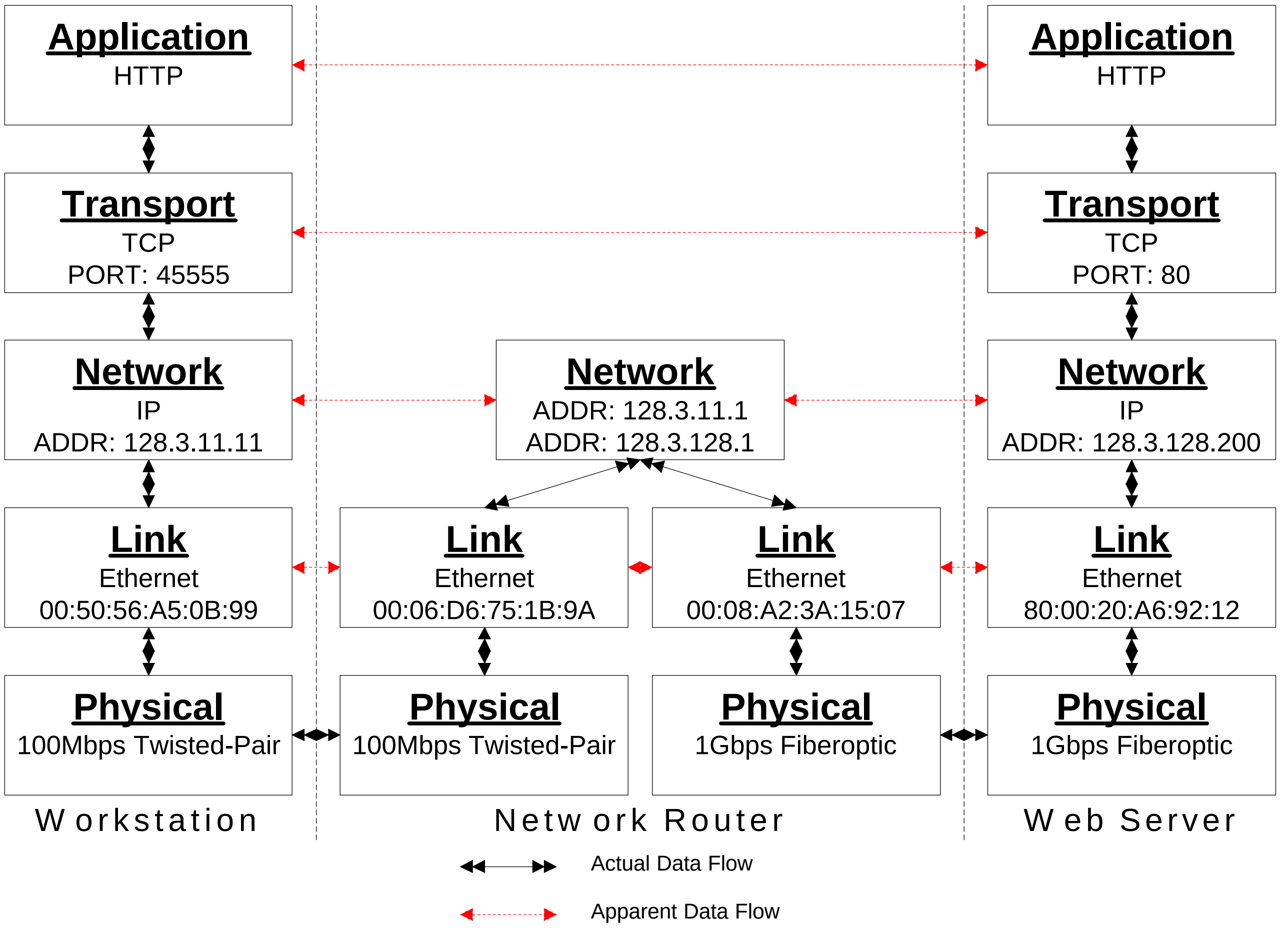
**Transport Layer**
TCP and UDP Protocols

**Network Layer**
IP Protocol, ICMP Messages

**Link Layer**
Ethernet Driver, Loopback Driver

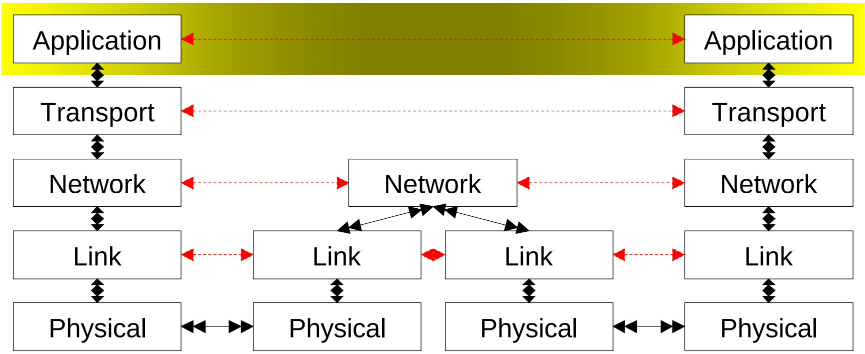
**Physical Layer**
Copper Wire, Fiberoptic Cable, Microwave, RF

## Sample HTTP Connection



# Internet Networking Application Layer

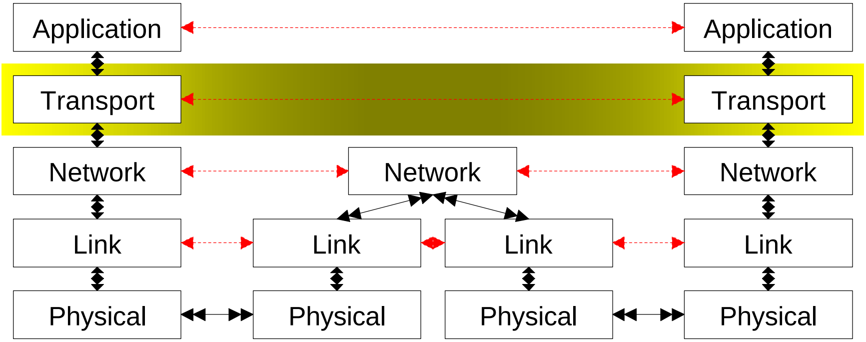
**Application Layer**
Client and Server Programs
(e.g. HTTP, SMTP)




- **End-points communicate with application-specific protocol:**
  - **e.g.: HTTP, SMTP, POP, IMAP, NTP, SSH**
- **Application asks transport layer to deal with establishing a connection, ensuring reliability (TCP only).**

# Internet Networking
# Transport Layer

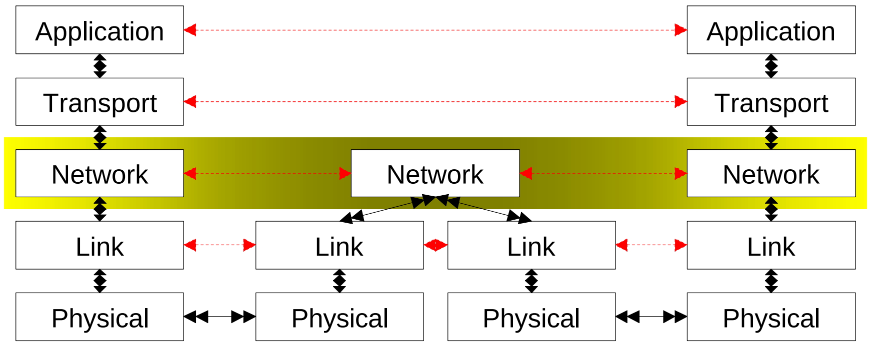
**Transport Layer**
TCP and UDP Protocols




- **End-points communicate with either TCP or UDP protocols.**
- **End-points are identified by port numbers (1 – 65535).**
- **A UDP connection is connectionless (neither packet arrival nor packet order is guaranteed).**
  - **e.g.: DNS, NFS, DHCP, video streaming**
- **A TCP connection is connection-oriented (if the connection is establish, packet arrival and order are guaranteed).**
  - **e.g.: HTTP, Telnet, SMTP, SSH**

# Internet Networking
# Network Layer

**<u>Network Layer</u>**
IP Protocol, ICMP Messages

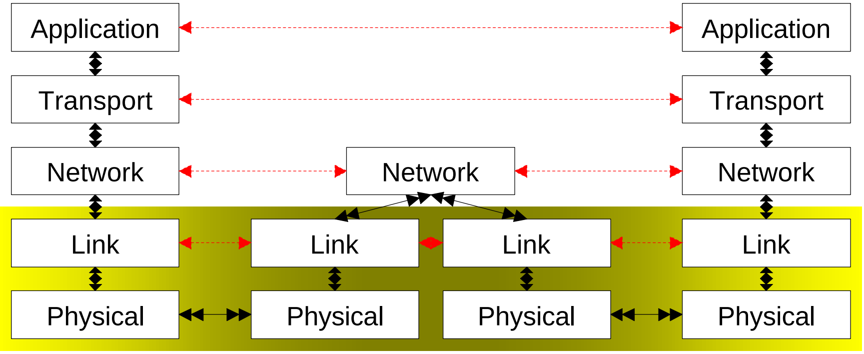



- **Network layer provides global host identification via internet-wide unique IP addresses.**
- **Also provides routing functionality.**
- **End-points are aware of intermediate “hops”.**
- **ICMP packets provide control and status messages (sent between IP layers, not applications).**

# Internet Networking
# Link & Physical Layers

**Link Layer**
Ethernet Driver,
Loopback Driver

**Physical Layer**
Copper Wire, Fiberoptic
Cable, Microwave, RF




- **It's not always easy to separate these into 2 layers – they are sometimes intricately linked.**
- **Link level – the lowest level of the protocol stack**
  - **—Each host adapter only knows about other hosts using the same protocol on the wire.**
  - **—Protocols: Ethernet, ATM, Token Ring, FDDI**
- **Physical level – the actual media used to send signals between hosts.**
  - **—e.g.: Twisted-pair copper, fiber, microwave, RF**

# Internet Networking
# Sample Connection Revisited

## Protocol Stack

**Application Layer**
Client and Server Programs (e.g. HTTP, SMTP)

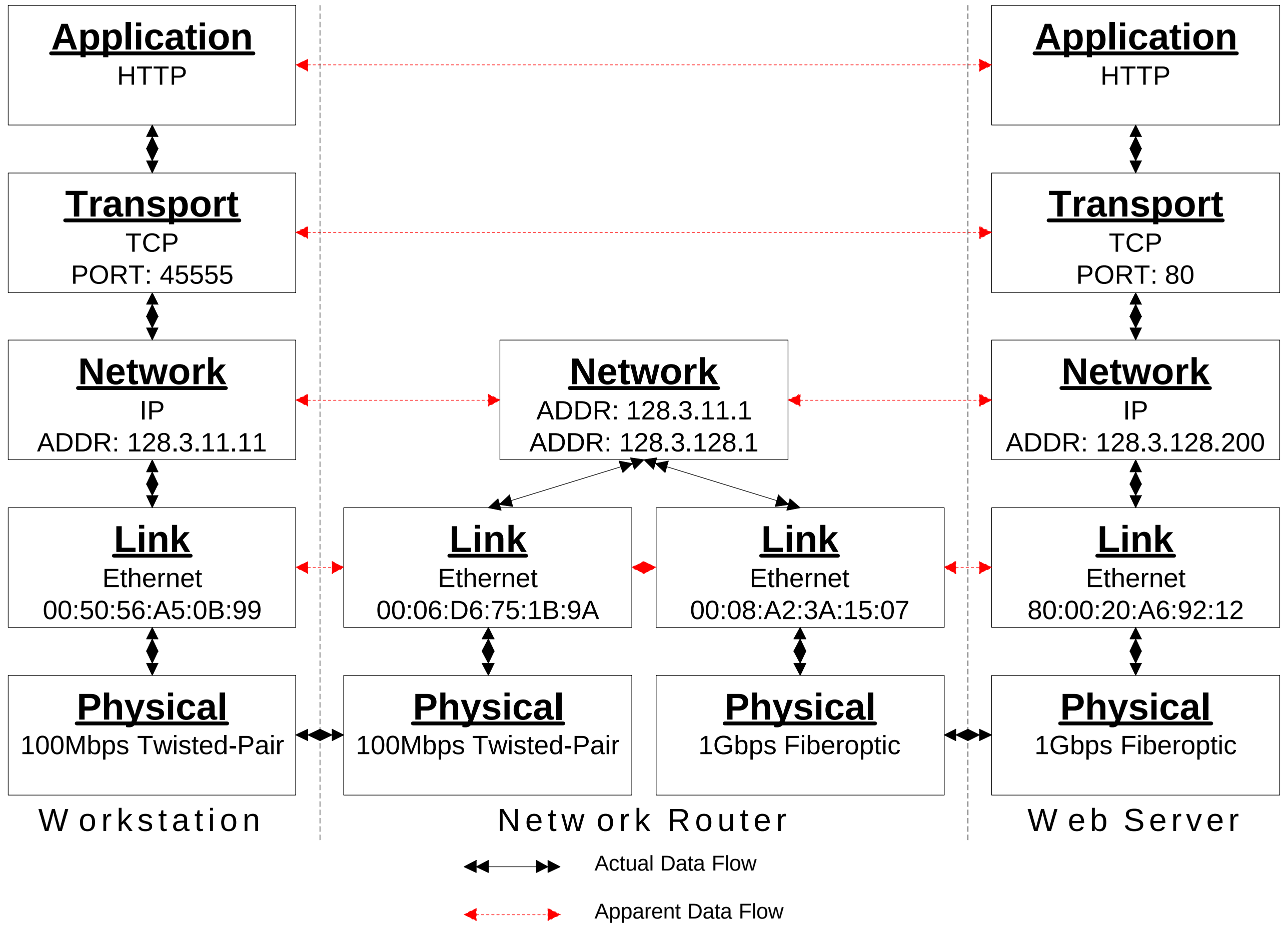
**Transport Layer**
TCP and UDP Protocols

**Network Layer**
IP Protocol, ICMP Messages

**Link Layer**
Ethernet Driver, Loopback Driver

**Physical Layer**
Copper Wire, Fiberoptic Cable, Microwave, RF

## Sample HTTP Connection



# Internet Networking
# Internet Control Message Protocol

- **ICMP packets are typically used to test if a host is alive, report errors and pass routing information.**
- **An ICMP packet is sent to a particular IP address, not to a port. It is handled by the network layer.**
- **ICMP packets are identified by type and, in some cases, codes (sub-type). (see RFC 1700 for list)**
- **ICMP packets are raw datagrams – they are unreliable.**
- **Most common user usage would be ping.**
  - **Host A sends an ICMP echo-request (type 8) packet to Host B.**
  - **Host B responds with an ICMP echo-reply (type 0) packet send to Host A.**

# Internet Networking
# Internet Control Message Protocol

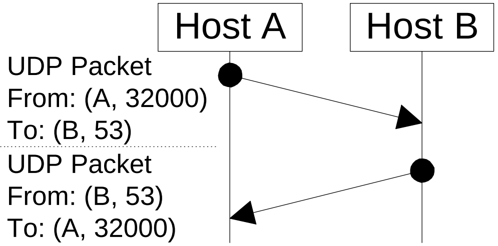
- Another common ICMP type is destination-unreachable (type 3).
  - There are many different sub-types, the most common are host unreachable and port unreachable.
  - You see port unreachable ICMP packets as a result of trying to connect to a port (normally UDP) that has no listening process.
- We care about ICMP packets because some are necessary and some can be misused for DoS attacks or worse – we need to weed out the ones we don’t want.

# Internet Networking
# User Datagram Protocol

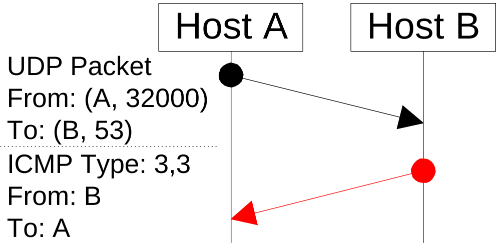
- **UDP packets are connectionless, unreliable.**
- **Since no *connection* is created, UDP is efficient.**
- **A UDP packet is sent from a source (IP address, port) to a destination (IP address, port).**
- **There is no acknowledgement required; the application protocol must implement its own acknowledgement protocol if it is so desired.**
- **At the destination host, any of these may happen:**
  - **—a UDP reply may be sent by an application**
  - **—a port-unreachable ICMP may sent by the OS**
  - **—nothing may be sent in reply**
- **Note that it is hard to tell if a UDP port is open, filtered, or if the host is not responding.**

# Internet Networking
# User Datagram Protocol

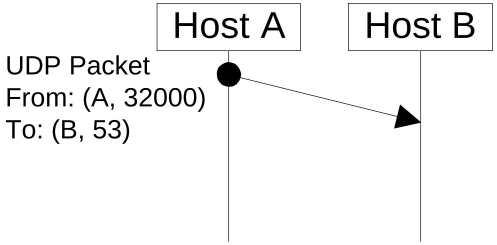
- **UDP PORT OPEN: A DNS request sent to host B receives a properly formatted reply.**




- **UDP PORT CLOSED: A DNS request sent to host B receives a port-unreachable (type 3, code 3) ICMP packet.**




- **UDP PORT STATE UNKNOWN: No reply is received; either host B is down, the DNS server choose not to respond, or port is filtered.**

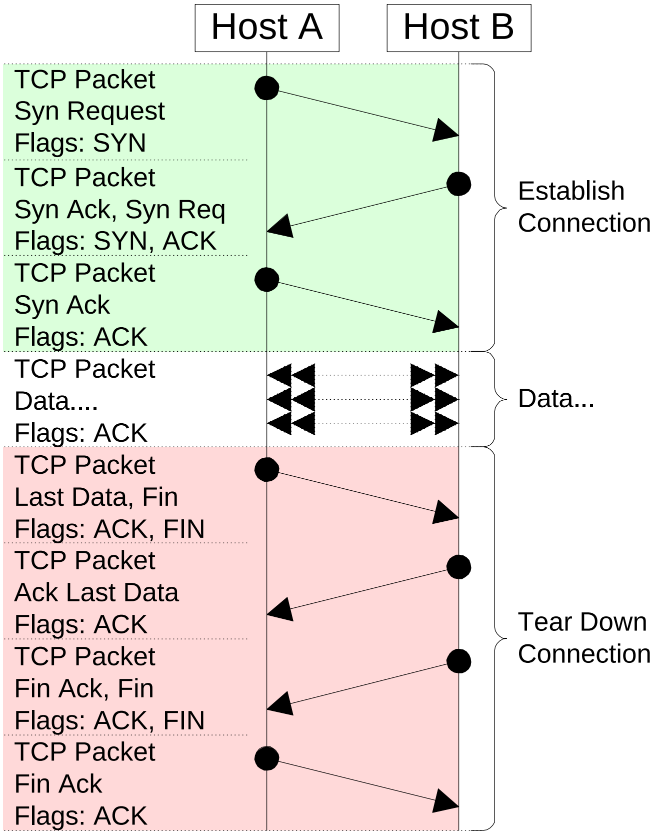


# Internet Networking
# Transmission Control Protocol

- **TCP packets are connection-oriented, reliable, and order-preserving.**
- **Before any data is sent over TCP, a *connection* is created between the two ends.**
- **A TCP connection is established between a source (IP address, port) and a destination (IP address, port). (Note: TCP and UDP ports are distinct.)**
- **The TCP transport layer ensures reliability and order – the application can assume reliability.**
- **Connections are established using a 3-way handshaking procedure.**

# Internet Networking Transmission Control Protocol

- **Diagram shows the 3-way handshaking for both establishing a connection and tearing down a connection.**
- **Understanding TCP header flags is very important when designing firewall rules.**
- **Many attacks involve using TCP packets with improper flags.**
- **Note a TCP packet with no flags set is *never* a valid packet.**



# Internet Networking
# Transmission Control Protocol

- **If a TCP packet arrives that apparently is not intended for the current session, a RST packet is sent in reply in order to *reset* the connection.**
- **This is what happens when a SYN packet is sent to try to open a connection to a closed port.  (Port unreachable ICMP will also get the job done, but RFC 793 specifies a RST TCP packet in this case.)**
- **Unlike UDP traffic, no response to a TCP packet is an unexpected condition resulting in a timeout.**
  - **If the host is up, it indicates a network error or the fact that the port is filtered.**
- **TCP port scans more accurately reveal open ports than UDP scans because a packet sent to an open TCP port *must* result in a reply.**

# Internet Networking Miscellaneous

- **Connections to well-known services are established by connecting to well-known published ports.**
- **UNIX hosts identify well-known ports via `/etc/services`.**
- **The IP header is self-identifying – an 8-bit *protocol* field identifies the protocol for this packet.**
- **UNIX hosts identify protocol via `/etc/protocols`.**
  - **It is sometimes useful to know that ICMP=1, TCP=6, and UDP=17; it shows up occasionally when dealing with iptables.**

# What is a Host-Based Firewall?

- In this context, a host-based firewall refers to a set of rules on an individual host that defines exactly what network traffic will be permitted in and out of that host.
- It is a packet filter that, based on certain matching criteria, decides on a per-packet basis whether to allow the packet through or drop the packet.
- It is normally implemented as part of the operating system at the network and transport layers of the protocol stack.
- In Linux, it is implemented with kernel modules and user-space tools.

# Why Should I Use a Host-Based Firewall?

- Security and Privacy
  - Port scans and exploits are frequent on machines connected to the Internet.
  - The less an *outsider* can determine about a host, the harder it is to compromise.
- A daemon for a required service may not be configurable to listen only to 127.0.0.1.
  - lpd – required even for local printing.
  - ntpd, bind – there are limited configuration options to restrict who they will talk to and these daemons often have exploits.
- Applications may listen on ports without you knowing about it – they may have vulnerabilities.
  - e.g.: X and many X/KDE/Gnome related applications.

# Why Should I Use a Host-Based Firewall?

- Even if an application or service provides host-based access restrictions (e.g. TCP wrappers) a connection is established for a short time.
  - For example, even if bind is configured to not respond to certain requests, it still receives the request. If bind has a buffer overflow vulnerability, it still may be susceptible.
- If the ports for these services are filtered, the packets are stopped in the network layer and are not passed up to the application or daemon.
- This can also help reduce the effects of being flooded with bad packets since it is more efficient to discard the packets as early as possible.

# Why Should I Use a Host-Based Firewall?

- **Should I used a host-based firewall even if our corporate network is behind a firewall?**
  - **—Here are some reasons:**
    - **If some machine inside the corporate firewall gets compromised, the corporate firewall is useless.**
    - **The corporate firewall may be more permissive than is required for your host.**
      - It's hard to make a ruleset for a corporate firewall that is right for every host on the network.
      - It should be easier to determine what traffic should and should not be allowed for your host.
    - **Protects your host from possible mischievous behavior from inside the organization.**

# Firewall Basics

- A packet-filtering firewall will, on a per-packet basis, attempt to match the packet with one or more rules in a ruleset.
- The match criteria could be anything that the firewall code supports.  Some of the more common criteria are:
  - Source or destination host or net IP addresses
  - Protocol, optionally including port(s) (for TCP, UDP) or type and code (for ICMP)
  - TCP header flags (e.g., SYN, RST, ACK…)
  - MAC (hardware) address(es)
  - Network interface
  - Packet size
  - Frequency (rate-limiting)

# Firewall Basics

- When a packet matches a rule, generally one of three actions is taken:
  - — The packet is accepted and passed in or out.
  - — The packet is dropped as if it never existed (the packet is “blackholed”).
  - — Some kind of rejection notification is sent to the source of the packet (like a port unreachable ICMP or a TCP RST packet).
- Most packet filters have separate rulesets for:
  - — Input packets – those coming into the host.
  - — Output packets – those leaving the host.
  - — Forwarded packets – those that arrive at the host on one adapter destined for a host on a different adapter (if the host is routing).

# Stateful (Dynamic) Packet Filtering

- **All of the criteria described so far are stateless – they don't depend upon the acceptance or denial of packets in the past.**
  - **—A firewall is termed a *static* packet filter if only stateless criteria are used (e.g., ipfw, ipchains).**
- **There are many cases when remembering how a previous packet was handled is useful in determining how to handle future packets.**
  - **—If you want to limit incoming TCP packets to responses from TCP connections *initiated* from the host, you have two choices:**
    - **Accept all incoming TCP packets except those that contain *only* the SYN flag; drop all SYN packets.**

# Stateful (Dynamic) Packet Filtering

  - Accept all incoming TCP packets that have the source (IP, port) and destination (IP, port) of a successful SYN packet sent out with the opposite end-points.
- Stateful or dynamic packet filters maintain a state table of accepted packets.
- Entries in this state table can be used as matching criteria for other rules.
- TCP entries in state table will exist for the duration of the connection and are removed when the connection is closed or some large time limit expires.
- Most dynamic packet filters keep state on protocols that are stateless (e.g., UDP and ICMP).

# Stateful (Dynamic) Packet Filtering

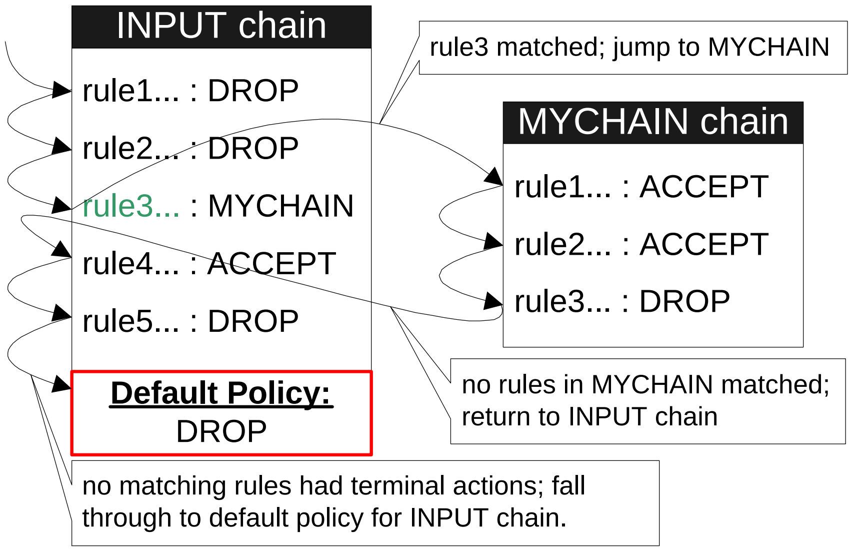
- **UDP entries in the state table have a relatively short timeout since there is no notion of connection with UDP packets.**
- **Stateful inspection of packets allows for a simplified and optimized ruleset.**
  - **Input rulesets only need explicit allow rules for services running on the host (e.g., web server).**
  - **By putting the rule that matches packets related to the state table entries near the head of the ruleset, traversal of most of the ruleset is avoided.**
- **IPTables, implemented in Linux since the 2.4 kernel, is a stateful packet filter.**

# IPTables Basics
# Rules, Chains, and Actions

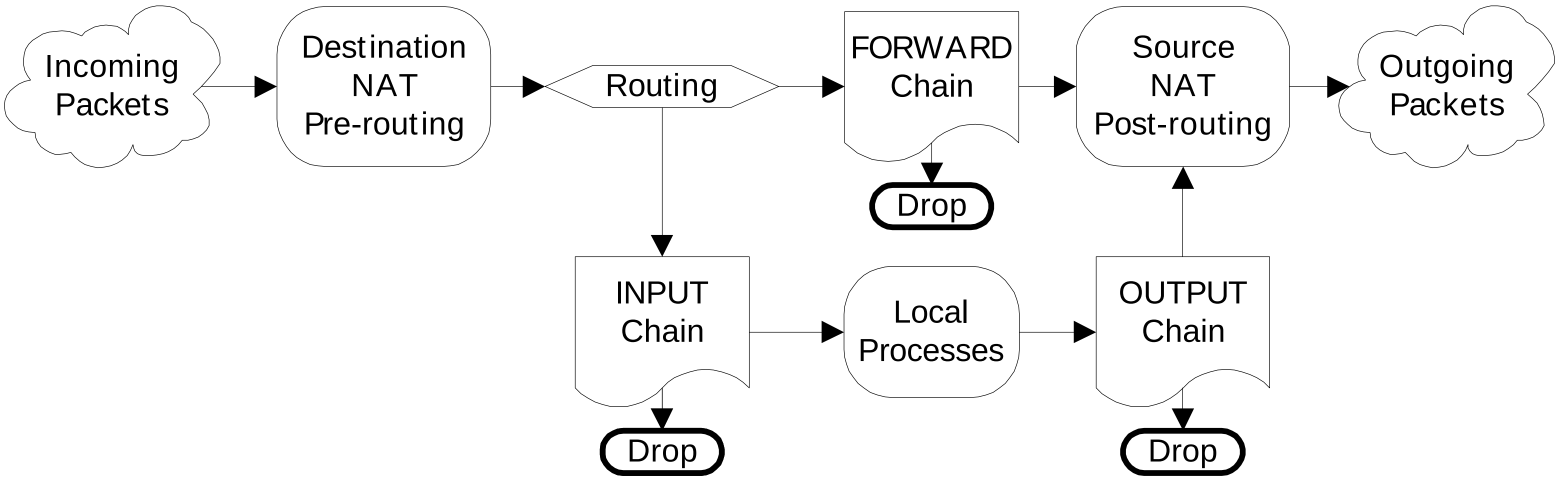
- IPTables organizes rules into *chains*.
- When a packet arrives at a chain to be matched, the rules are processed one at a time from the first rule to the last.
- When a match occurs, an action (or target) specified by the rule is taken.
  - —The action may be *terminal*; i.e., drop the packet or accept the packet.
  - —The action may not be terminal; i.e., log the packet or *jump* to another chain.
- If a packet matches a terminal action, processing stops there; rule order is first-come, first-served.

# IPTables Basics
# Rules, Chains, and Actions

- **There are three built-in "filter" chains in IPTables: `INPUT`, `OUTPUT`, and `FORWARD`.**
- **If processing reaches the end of a built-in chain, a *default policy* indicates what will become of the packet (either it is accepted or dropped).**
- **If processing reaches the end of any other (user-defined) chain, it will return to the chain from whence it came and continue where it left off.**



# IPTables Basics
# Rules, Chains, and Actions




Netfilter packet traversal, including NAT routing.
(Figure based on "Linux 2.4 Packet Filtering HOWTO" and "Linux 2.4 NAT HOWTO".)

- **All incoming packets, including those on the loopback interface, must make it through the `INPUT` chain.**
- **As such, most of the rules you will write for a stand-alone host-based filewall will be part of the `INPUT` chain.**
- **All packets get filtered once (except loopback packets, which are filtered twice).**
- **We won't discuss NAT here – the NAT boxes above will essentially be transparent.**

# IPTables Basics
# Rules, Chains, and Actions

- As indicated before, there are 3 built-in "filter" table chains: `INPUT`, `OUTPUT`, and `FORWARD`.
- However, IPTables has two other packet matching tables – one used for NAT, the other for specialized packet alterations (called "mangle").
- The "nat" table has 3 built-in chains: `PREROUTING`, `POSTROUTING`, and `OUTPUT`.
- The "mangle" table has 2 built-in chains: `PREROUTING`, and `OUTPUT`.
- We won't discuss these except to note that if you are not routing or doing NAT on your host, you'll want to make sure the default policy for the built-in "nat" chains is ACCEPT.

# IPTables Basics
# Rules, Chains, and Actions

- IPTables is built on modules – both kernel modules (support for the main `ip_filter` module) and user-space modules (for the `iptables` program).
- There are modules to provide support for matching criteria and modules that implement targets (actions).
- The only built-in targets are `ACCEPT` and `DROP`.
- Other useful targets are:
  - user-defined chains.
  - `LOG` – logs the packet to syslog (kernel facility). (non-terminal target – processing continues after logging.)
  - `REJECT` – drops the packet and sends a port unreachable ICMP or RST TCP reply.
- There are interesting and experimental matching and target modules; since they are easily written as modules, I expect the open source community to be very creative here.

# IPTables Basics
# Command Syntax

- **All user interaction with the netfilter kernel modules is done through the user-space tool called `iptables`.**
- **`iptables-save` and `iptables-restore` are additional tools that will dump and restore the current netfilter tables via `iptables`. These tools are used by RedHat's default `/etc/init.d` script.**
- **A rule is constructed by specifying a command, a matching criteria and a target:**

Matching Criteria:
Incoming on interface eth0

Target:
Drop packet

```
iptables -A INPUT -i eth0 -s 127.0.0.0/8 -j DROP
```

Command:
Append to INPUT chain

Matching Criteria:
Source IP=127.0.0.0/8

**Drop incoming packets on eth0 that have a "loopback" source address.**

# IPTables Basics Command Syntax

## Basic commands:

| OPTION | DESCRIPTION |
|---|---|
| `-L [<chain>] [-n] [-v] [-x] [--line-numbers]` | List rules; `-n` = numeric; `-v` = verbose; `-x` = exact counter values; `--line-numbers` = rule number |
| `-N <chain>` | Create a new user-defined chain. |
| `-F [<chain>]` | Flushes the chain, or all chains if none specified. |
| `-X [<chain>]` | Delete user-defined chain, or all if none specified. |
| `-P <chain> <policy>` | Sets the default policy for the built-in chain. |
| `-Z` | Resets the packet and byte counters. |
| `-A <chain>` *{rule…}* | Append the rule to the end of the chain. |
| `-I <chain> [<rule num>]` *{rule…}* | Insert the rule before the specified rule number (default rule number is 1 – inserts at chain head). |
| `-D <chain> <rule num>` | Deletes the rule at the given position number. |
| `-h \| <some command> -h` | Lists iptables commands and options. If proceeded by a command, list syntax and options for that command. (e.g.: `-m limit -h`) |

# IPTables Basics
# Command Syntax

## Basic matching criteria and other options:

| OPTION | DESCRIPTION |
| --- | --- |
| `-i [!] <interface>` | For incoming packets, specify the interface name that the rule applies to. [INPUT, FORWARD] |
| `-o [!] <interface>` | For outgoing packets, specify the interface name that the rule applies to. [OUTPUT, FORWARD] |
| `-p [!] <proto>` | Specifies IP protocol the rule applies to; can be `tcp`, udp, `icmp`, `all` or any numeric value. |
| `-s [!] <addr>[/<mask>]` | Specifies host or network source address in the packet’s IP header. |
| `-d [!] <addr>[/<mask>]` | Specifies host or network destination address in the packet’s IP header. |
| `-m <match module> …` | Use extended matching module; each module may have its own set of extra options. |
| `[!] -f \| --fragment` | Match second or further fragments only. |
| `-t <table>` | Specifies packet matching table command should operate on: `filter` (default), `nat`, or `mangle`. |

# IPTables Basics
# Command Syntax

## TCP, UDP, and ICMP additional matching criteria:

| -p tcp OPTIONS | DESCRIPTION |
|---|---|
| --sport [!] <port>[:<port>] | Specifies source port or range of ports. |
| --dport [!] <port>[:<port>] | Specifies destination port or range of ports. |
| --tcp-flags [!] <mask>[,<mask>]* <set>[,<set>]* | Tests the bits in the mask list. The bits in the set list must be set for this option to match. |
| [!] --syn | The SYN flag must be set. |

| -p udp OPTIONS | DESCRIPTION |
|---|---|
| --sport [!] <port>[:<port>] | Specifies source port or range of ports. |
| --dport [!] <port>[:<port>] | Specifies destination port or range of ports. |

| -p icmp OPTIONS | DESCRIPTION |
|---|---|
| --icmp-type [!] <type> | Specifies ICMP type name or number. |

# IPTables Basics Command Syntax

## State matching criteria:

| -m state OPTIONS | DESCRIPTION |
|---|---|
| `--state <state>[,<state>]*` | Matches if the connection state is one in the list. `state` = `NEW`, `ESTABLISHED`, `RELATED`, `INVALID`. |

| STATE | DESCRIPTION |
|---|---|
| `NEW` | Matches if the packet starts a new connection or is otherwise associated with a connection that has not been seen in both directions. |
| `ESTABLISHED` | Matches if the packet is associated with a connection that has seen packets in both directions. This refers to ongoing TCP ACK packets after the connection is established; subsequent UDP datagrams exchanged between the same hosts and ports; certain ICMP pairs, like `echo-reply` in response to a previous `echo-request`. |
| `RELATED` | Matches if the packet is starting a new connection but is associated with an existing connection, such as an FTP data transfer (requires FTP conntrack module) or an ICMP error. |
| `INVALID` | Matches if the packet is associated with no known connection. |

# IPTables Basics Command Syntax

## Multiport, mac, and limit matching criteria:

| -m multiport OPTIONS | DESCRIPTION |
|---|---|
| `--source-port <port>[,<port>]*` | Specifies source port(s). |
| `--destination-port <port>[,<port>]*` | Specifies destination port(s). |
| `--port <port>[,<port>]*` | Source and destination ports are equal and they match a port in the list |
| **-m mac OPTIONS** | **DESCRIPTION** |
| `--mac-source [!] <addr>` | Matches the ethernet hardware source address. |
| **-m limit OPTIONS** | **DESCRIPTION** |
| `--limit <rate>` | Maximum number of packets to match within the given time frame (eg, 3/second) (default: 3/hour). |
| `--limit-burst <number>` | Maximum number of initial packets to match before applying the limit (default: 5). |

# IPTables Basics Command Syntax

## Targets:

| TARGET | DESCRIPTION |
|---|---|
| `-j ACCEPT` | Packet is accepted and passed through. |
| `-j DROP` | Packet is dropped as if it was never sent. |
| `-j <chain>` | Send packet through <chain>. |
| `-j REJECT` | Drops packet but sends a reply of some sort (by default, an ICMP `port-unreachable` message). |
| `-j LOG` | Logs the packet to syslog, kernel facility. Processing continues to next rule in chain after the packet is logged. |
| `-j MIRROR` | Inverts source and destination fields in IP header and retransmits the packet. (experimental, demo) |

# IPTables Basics Command Syntax

## REJECT and LOG target options:

| -j REJECT OPTIONS | DESCRIPTION |
|---|---|
| `--reject-with <ICMP type 3>` | Specify other ICMP type 3 (unreachable) message; default is `port-unreachable`. |
| `--reject-with tcp-reset` | Incoming TCP packets can be rejected with the RFC specified TCP RST packet. |
| `--reject-with echo-reply` | Can be used to return an `echo-reply` to an `echo-request` without forwarding to actual target host. |

| -j LOG OPTIONS | DESCRIPTION |
|---|---|
| `--log-level <syslog lvl>` | Syslog log level (default: warn (4)). |
| `--log-prefix <"descriptive string">` | Prepend the quoted string at the start of the log message for this rule. |
| `--log-ip-options` | Includes any IP header options in log. |
| `--log-tcp-sequence` | Includes the TCP sequence number in log. |
| `--log-tcp-options` | Includes any TCP header options in log. |

# Building a Usable Stand-Alone Firewall Ruleset

- When building up a ruleset, you need to determine:
  - What services do you need to run (ssh, http)?
  - Do any of your services require RPC?
  - Do you need to share files with NFS (either as a client or server)? (NFS uses RPC.)
  - How restrictive do you want to be?
  - How much information do you want to share?
  - Does your organization require any specific ports, services, or behavior (like not dropping packets)?
  - Are you part of an NIS domain?
  - Do you use DHCP for a dynamic IP address?

# Building a Usable Stand-Alone Firewall Ruleset

- **Assumptions for this example:**
  - —We want to be restrictive. We can add things later.
  - —We want privacy.
  - —We don't use NFS, NIS, or any other RPC services.
  - —We have a fixed IP address (no DHCP).
  - —We want ssh enabled from anywhere.
  - —We want httpd enabled from "on-site".
  - —We want to log dropped packets.
  - —Our organization doesn't have any special restrictions.

# Building a Usable Stand-Alone Firewall Ruleset

- **Our host:**
  - —**IP address: 12.5.7.15**
  - —**Netmask: 255.255.255.0 (24)**
  - —**Gateway: 12.5.7.1**
  - —**“on-site” networks: 12.5.7.0/24, 12.5.8.0/24**

# Building a Usable Stand-Alone Firewall Ruleset (Example 1/6)

- **First off, remove pre-existing rules and chains:**

```
iptables -F                  # flushes all "filter" chains
iptables -X                  # deletes all user-defined "filter" chains
iptables -t nat -F           # flushes all "nat" chains
iptables -t mangle -F        # flushes all "mangle" chains
```

- **Then set our default policies:**

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT
iptables -t nat -P PREROUTING ACCEPT
iptables -t nat -P POSTROUTING ACCEPT
iptables -t nat -P OUTPUT ACCEPT
```

- **Next we are going to create our chains:**

```
iptables -N EXT-IN           # Packets not from “on-site” go here
iptables -N ONSITE-IN        # Packets from “on-site” go here
iptables -N SPOOF            # Checks for spoofed IP addresses
iptables -N SCAN             # Checks for bad TCP flags (funny port scans)
iptables -N ICMP             # Narrows down on the ICMP message we want
```

# Building a Usable Stand-Alone Firewall Ruleset (Example 2/6)

- **Since logging and dropping a packet actually takes two rules, we create special chains for logging and dropping packets.  We also create a chain with different description strings so that we can differentiate them in the log file.**

```
iptables -N SPOOFDROP                    # Where SPOOF packets are logged and dropped
iptables -A SPOOFDROP -j LOG --log-prefix "SPOOF DROP " --log-
   ip-options --log-tcp-options --log-tcp-sequence
iptables -A SPOOFDROP -j DROP
```

- **We will create chains with the exact same rules (rules are quite uninteresting and very small…):**

```
iptables -N SCANDROP                     # Where SCAN packets are logged and dropped
iptables -A SCANDROP -j LOG --log-prefix "SCAN DROP " --log-ip-options --log-tcp-options --log-tcp-sequence
iptables -A SCANDROP -j DROP
iptables -N ICMPDROP                     # Where bad ICMP packets are logged and dropped
iptables -A ICMPDROP -j LOG --log-prefix "ICMP DROP " --log-ip-options --log-tcp-options --log-tcp-sequence
iptables -A ICMPDROP -j DROP
iptables -N POLICYLOG                    # Log "POLICY" drop before POLICY is enforced
iptables -A POLICYLOG -j LOG --log-prefix "POLICY DROP " --log-ip-options --log-tcp-options --log-tcp-sequence
iptables -N INVALDROP                    # Where INVALID state pkts are logged and dropped
iptables -A INVALDROP -j LOG --log-prefix "INVALID DROP " --log-ip-options --log-tcp-options --log-tcp-sequence
iptables -A INVALDROP -j DROP
iptables -N EXTDROP                      # Leftover EXT-IN pkts are logged and dropped
iptables -A EXTDROP -j LOG --log-prefix “EXT-IN DROP " --log-ip-options --log-tcp-options --log-tcp-sequence
iptables -A EXTDROP -j DROP
```

# Building a Usable Stand-Alone Firewall Ruleset (Example 3/6)

- **We begin by defining the INPUT chain:**

```
iptables -A INPUT -i lo -j ACCEPT        # Accept all incoming packets on loopback
iptables -A INPUT -p tcp -j SCAN         # Check all TCP packets for bad flags
iptables -A INPUT -i eth0 -j SPOOF       # Check all eth0 packets for spoofing
# Accept all established/related packets
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
   ACCEPT
# Log and drop INVALID state packets
iptables -A INPUT -m state --state INVALID -j INVALDROP
# Send on-site packets to ONSITE-IN
iptables -A INPUT -s 12.5.7.0/24 -d 12.5.7.15 -i eth0 -j
   ONSITE-IN
iptables -A INPUT -s 12.5.8.0/24 -d 12.5.7.15 -i eth0 -j
   ONSITE-IN
iptables -A INPUT -i eth0 -j EXT-IN      # Send all other eth0 packets to EXT-IN
iptables -A INPUT -p icmp -i eth0 -j ICMP       # Check for bad ICMP
iptables -A INPUT -j POLICYLOG           # Log that it is to be dropped by policy
```

# Building a Usable Stand-Alone Firewall Ruleset (Example 4/6)

- **Now define the IP address SPOOF chain (remember this is used only for packets that arrive on the eth0 interface):**

```
iptables -A SPOOF -s 12.5.7.15 -j SPOOFDROP                # Our IP as src
iptables -A SPOOF -s 10.0.0.0/8 -j SPOOFDROP               # Class A private src
iptables -A SPOOF -s 172.16.0.0/12 -j SPOOFDROP            # Class B private src
iptables -A SPOOF -s 192.168.0.0/16 -j SPOOFDROP           # Class C private src
iptables -A SPOOF -s 224.0.0.0/4 -j SPOOFDROP              # Multicast src
iptables -A SPOOF –p ! udp –d 224.0.0.0/4 -j SPOOFDROP # !udp mcast
iptables -A SPOOF -s 240.0.0.0/5 -j SPOOFDROP              # Class E (reserved)
iptables -A SPOOF -s 127.0.0.0/8 -j SPOOFDROP              # loopback
iptables -A SPOOF -s 169.254.0.0/16 -j SPOOFDROP           # Link local networks
iptables -A SPOOF -s 192.0.2.0/24 -j SPOOFDROP             # TEST-NET
iptables -A SPOOF -s 255.255.255.255/32 -j SPOOFDROP # Bcast dest as src
iptables -A SPOOF -d 0.0.0.0/32 -j SPOOFDROP               # Bcast src as dest
```

- **Note that if your IP address is on a private subnet, you will have to modify these rules so you don’t drop them!**

# Building a Usable Stand-Alone Firewall Ruleset (Example 5/6)

- **Now define the SCAN chain that checks for bad TCP flags:**

```
iptables -A SCAN -p tcp --tcp-flags ALL NONE -j SCANDROP
iptables -A SCAN -p tcp --tcp-flags SYN,FIN SYN,FIN -j SCANDROP
iptables -A SCAN -p tcp --tcp-flags SYN,RST SYN,RST -j SCANDROP
iptables -A SCAN -p tcp --tcp-flags FIN,RST FIN,RST -j SCANDROP
iptables -A SCAN -p tcp --tcp-flags ACK,FIN FIN -j SCANDROP
iptables -A SCAN -p tcp --tcp-flags ACK,PSH PSH -j SCANDROP
iptables -A SCAN -p tcp --tcp-flags ACK,URG URG -j SCANDROP
```

- **Check for ICMP packets we want and drop the rest:**

```
iptables -A ICMP --fragment -p icmp -j ICMPDROP
iptables -A ICMP -p icmp --icmp-type source-quench -j ACCEPT
iptables -A ICMP -p icmp --icmp-type parameter-problem -j ACCEPT
iptables -A ICMP -p icmp --icmp-type destination-unreachable -j
   ACCEPT
iptables -A ICMP -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A ICMP -p icmp --icmp-type echo-request -j ACCEPT
iptables -A ICMP -j ICMPDROP
```

- **Note that** `source-quench` **can be used for DoS attack but are still sometimes used for flow-control on LANs.**

# Building a Usable Stand-Alone Firewall Ruleset (Example 6/6)

- **Now define rules for packets from "on-site":**

```
# Accept all HTTP (SYN) packets from on site; the rest of the traffic is handled by ESTABLISHED rule.
iptables -A ONSITE-IN -p tcp --dport 80 --syn -m state --state
   NEW -j ACCEPT
```

- **Finally, we finish with the rules for other external packets:**

```
iptables -A EXT-IN -p tcp --dport 22 ––syn -m state --state NEW -
   j ACCEPT
iptables -A EXT-IN -p tcp --dport 113 -j REJECT --reject-with
   tcp-reset
iptables -A EXT-IN -p tcp -j EXTDROP
iptables -A EXT-IN -p udp -j EXTDROP
```

- **The port 113 REJECT is to allow ident requests to fail nicely.**
- **Even though the default policy would drop the remaining tcp and udp packets, I do it explicitly so it is logged as an external packet that I didn't want to accept.**

# Building a Usable Stand-Alone Firewall Ruleset

- Why doesn't my FTP client work?
  - —Passive FTP should work fine here because of the ESTABLISHED rule in the INPUT chain.
  - —Active (PORT) FTP, however, will not work since that requires the creation of a data channel from the server back to the client.
  - —Thankfully, you can load a module that fixes this – it adds functionality to the RELATED state that allows an active FTP data connection back from the server.

```
# modprobe ip_conntrack_ftp
```

  - —This needs to be done sometime before attempting an active FTP connection.

# Building a Usable Stand-Alone Firewall Ruleset

- **Yeah, but how do I allow an FTP server even though I know it is a security risk?**
  - **Simply adding a rule that allows the FTP command channel port through will work for active FTP (this needs to go before the EXTDROP rules in the EXT-IN chain).**

```
iptables -A EXT-IN -p tcp --dport 21 -m state --state
   NEW --syn -j ACCEPT
```

  - **However, now passive FTP is a problem since the server sends the PORT command.**
  - **Again, if the `ip_conntrack_ftp` module is loaded before a passive connection takes place, the RELATED state will allow this to work.**

# Diagnosing Firewall Problems

- **If you have an application or service that is not working when the firewall is enabled:**
  - **Use the logs generated by netfilter:**

    `# tail –f /var/log/messages | grep …`
  - **Use tcpdump (see man page for tcpdump(8)).**
  - **Use lsof to show you processes that have ports open:**

    `# lsof –i` *`(may append tcp or udp to restrict)`*
  - **Check the packet/byte counts that iptables maintains for your rules to see which rules are being hit:**

    `# iptables –L [<chain>] –v -n`
  - **Use nmap to port scan your host to see what snoopers will see:** `http://www.insecure.org/nmap`
  - **Look at the connection tracking table to debug stateful inspection problems:**

    `# cat /proc/net/ip_conntrack`

# Diagnosing Firewall Problems Notes about RPC

- **Applications and services that use RPC will have to have special provisions made to enable them through the firewall.**
- **This example ruleset will function work with NFS clients, but not completely since** `rpc.statd` **cannot be seen from outside of the firewall.**
- **If you need to use RPC services, you will need to write a script that takes the output of** `rpcinfo -p` **and generates rules dynamically. This should be run at the end of the booting process.**

# References


- Ziegler, Robert L. Linux Firewalls, Second Edition. Indianapolis, Indiana: New Riders Publishing, 2002. ISBN: 0-7357-1099-6.
- Zwicky, Elizabeth D., et al. Building Internet Firewalls (2nd Edition). Sebastopol, CA: O'Reilly & Associates, 2000. ISBN: 1-5659-2871-7.

- Many great documents and howtos can be found at the netfilter documentation page: `http://netfilter.samba.org/documentation/index.html`
  - Networking Concepts HOWTO
  - Packet Filtering HOWTO
  - NAT HOWTO
  - "iptables connection tracking" Tutorial